9 Python

9.1 介绍

Python 脚本是 TouchDesigner 0.88 中最强大的功能之一。它能做非常复杂的操作，如迭代运算大量数据集合，自带的大量 Web 通信 API，能以一种超乎寻常的速度去控制和改变元件参数等等。

有一点需要注意， TouchDesigner 088 使用 Python 3.3.1。Python 3 在许多地方与 Python 2.7 不同，但是大部分问题都可以通过谷歌搜索得到解决。许多问题可能已经被提过，其解决方法可能只是添加一组小括号。

例如，在 Python 2.7，Print 函数可以不添加小括号，但在 Python 3 中会将抛出一个错误。下面的案例是在 Python 2.7 中使用 Print 方法。

print 'text'

在 Python 3 中使用对应的 Print 方法，小括号是必需的：

print ('text')

Python 中一个令人难以置信的功能是除了它的代码可读性外，它还有许多的外部库，其中一部分可以直接连接到最活跃的网站，如 Facebook、Twitter、Foursquare 等等。这类开放式实时数据驱动项目作为可视化项目和为客户演示的实时数据。

在后面的案例中会有相当一部分涉及到 Python 脚本语言，虽然不强制这么做，但还是强烈建议一定要花一些时间学习 Python 的入门教程。有许多 Python 的学习资源，其中一些是做游戏，它们中的大多数很容易找到并下载下来。每天花费 10-15 分钟，坚持几个星期，可以大幅提升 Python 的基础知识，以及拥有足以编写 TouchDesigner 脚本的能力。

9.2 Textport

在 TouchDesigner 中的 Textport 对于脚本使用扮演着几个重要的角色。有两种打开 Textport 的方法，第一种使用快捷键 “Alt+T” ，第二种在 TouchDesigner 窗口的顶部 “Dialogs” 菜单中选择 “Textport ” 。

第一种方法是自带的编辑器，就像我们在 Python 自带的 IDLE shell 编辑环境一样。例如，打开 Textport，输入下面代码，按回车确认。

print(2+2+2+2)

在 Python 中运行这句代码会打印出等式结果为 8，在 Textport 中按回车后同样会显示等式结果。这是因为 Textport 作为 Python 解释器运行。键入 Python 代码之后，它将会被执行，并返回结果。

对于在 TouchDesginer 077 中运行的脚本语言 tscript 可以作相同的运算。注意当 Textport 被打开时，第一次打印出来的信息是：

python >>

这是因为在 TouchDesigner 088，Textport 的默认解释器为 Python。要在 Textport 中运行 tscript，需要将 Textport 的模式从 Python 改为 tscript。点击 Textport 左上角的 Python Logo 标志，它将会变成字母“T”，在 Textport 的下一行将打印：

tscript ->

这就说明 Textport 目前处于 tscript 模式。在 Textport 中输入下面代码进行验证：

echo Hello!

与 Python 的 Print 函数相似，上面的 tscript 代码会在 Textport 中打印“Hello!”。

Textport 的另一个重要功能是作为 Python 的调试器。打开“Textport_1.toe”。

这一简单例子主要演示当运行 Python 脚本时如何结合 Textport 一起使用。打开 Textport，点击“Good Python”按钮，运行脚本，会在 Textport 中打印如下信息：

this will not error

现在点击“Bad Python”按钮，Textport 显示一些非常不同的信息：

File "/project1/chopexec1", line 11
Print(this will error)
^
SyntaxError: Invalid syntax

这是一个 Python 错误，主要因为一些代码被解释为无效的。学习阅读这些错误可以加速调试大型 Python 脚本。让我们检查这个错误的各个不同部分。

第一行指定错误出现于网络中的确切位置，以及哪一行脚本是被 Python 解释器认为是无效的。

File "/project1/chopexec1", line 11

在这个案例，错误的元件是“project1”组件内的“chopexe1”元件，Python 在脚本的第 11 行停止解释。

下面，Textport 打印的错误代码行：

Print(this will error)

多数时候，拼写错误和大小写错误可以轻易的通过这两行发现。在这种情况下，很明显要打印的字符串不在封闭的引号中。知道错误原因和出错位置，意味着问题可以很快得到解决。再来看看错误的最后一行：

SyntaxError: Invalid syntax

错误的最后一行是 Python 遇到的错误类型。关于不同错误类型的更详细信息可以在 Python 3.3 的离线文档中的“Errors and Exceptions”章节找到。“SyntaxErrror”是很常见的错误，会导致 Python 解释器遇到不正确的 Python 语法代码。如上所述，上面的代码是因为要打印的字符串丢失了引号。

在 Textport 中打印信息是脚本输出的一种调试方法。在大型项目中脚本随处可见，而且它们中有大都在后台执行任务，例如影片的预加载与卸载，它们的运行结果都很难被观察到。多数时候，脚本在不同网络中是并行运行的，这就很难观察到脚本的运行时间是否正常，以及判断它们动作发生的先后顺序。

在运行脚本时通过打印方法，许多零碎的信息会被打印出来，不易于分析脚本。这些信息片段可以简单作为被运行脚本的路径，或作为被改变值或改变的详细说明。

打开“Textport_2.toe”，在这个案例，有两个依次间隔运行的脚本序列。单击“Visual Sequence”按钮，通过将序列中按钮设置为 On 启动每个脚本的运行。这是为了更清楚看到这个序列的运行情况。那么如何从其他网络监视这个过程呢？

打开 Textport，单击“Textport Sequence”按钮。不同于第一个序列，这个序列把在 Textport 上打印信息作为每个脚本的运行。在网络中没有显示改变状态的按钮，但是有一些新的功能。第一，在项目中的任何位置都可以监视这些脚本。第二，在项目中的其他位置脚本都可以对比这些序列的运行时间。第三，这可能是最有用的，暂停项目的运行并将最近序列事件写入历史记录。

当调试拥有复杂逻辑的系统时，这段历史记录就变得绝对有价值。在一个拥有 30 个网络和 300 个脚本的项目中，当一系列动作失败却没有任何 Python 错误抛出时，没有一个有序的日志记录，就根本不可能跟踪调试。由于脚本变得越来越复杂，在脚本中轻易创建更多的这样伪检查点，如“Y 脚本的 X 部分正在运行中”。

9.3 规范

在 Python 中有一些约定俗成的东西在工作时应该使用。

关于 Python 语言的一些有趣东西构建在它可读与易用的思想上。在 Textport 或 Python 解释器中输入下面代码，可以看到 Python 的一个复活节彩蛋：

import this

Python 返回一首诗，名为“The Zen of Python”，由 Tim Peters 创作。下面是原文：

Python 之禅
赖勇浩翻译

优美胜于丑陋（Python 以编写优美的代码为目标）
明了胜于晦涩（优美的代码应当是明了的，命名规范，风格相似）
简洁胜于复杂（优美的代码应当是简洁的，不要有复杂的内部实现）
复杂胜于凌乱（如果复杂不可避免，那代码间也不能有难懂的关系，要保持接口简洁）
扁平胜于嵌套（优美的代码应当是扁平的，不能有太多的嵌套）
间隔胜于紧凑（优美的代码有适当的间隔，不要奢望一行代码解决问题）
可读性很重要（优美的代码是可读的）
即便假借特例的实用性之名，也不可违背这些规则（这些规则至高无上）
不要包容所有错误，除非你确定需要这样做（精准地捕获异常，不写 except:pass 风格的代码）

当存在多种可能，不要尝试去猜测
而是尽量找一种，最好是唯一一种明显的解决方案（如果不确定，就用穷举法）
虽然这并不容易，因为你不是 Python 之父（这里的 Dutch 是指 Guido ）
做也许好过不做，但不假思索就动手还不如不做（动手之前要细思量）
如果你无法向人描述你的方案，那肯定不是一个好方案；反之亦然（方案测评标准）
命名空间是一种绝妙的理念，我们应当多加利用（倡导与号召）

这首诗是许多 Python 开发者的座右铭。它的字里行间传达着 Python 的理想和开发的规范。
思考这一行：

'Explicit is better than implicit'
明了胜于晦涩（优美的代码应当是明了的，命名规范，风格相似）

这句话可以用到经常匆忙使用的事情上：命名。在 Python 与 TouchDesigner 中有许多不同地方都有对象之间通过名称互相引用。每一个开发者都要命名变量、方法，元件、网络等等。如果没有对元件仔细命名，不可能能知道每个元件的方法。类似的，本章节前面的部分已经证明，如果没有仔细命名变量名，阅读脚本会变成一件非常枯燥的任务。在 TouchDesigner 中当命名元件或变量时有两种格式：

第一种，每个单词之间使用下划线分割。如：
final_comp
stop_button
time_now

下划线分割使得名字容易阅读并快速理解。但有一些人不喜欢使用下划线，而是在不同的单词之间使用大写字母来区分单词。如：
finalComp
stopButton
timeNow

这两种方法都可以明确传达开发者的原始想法，促进合作。

9.4 注释

对于 Python 初学者，注释应该作为一种练习，不能被忽视。它很简单，只需要几秒钟来说明那几行代码的作用。在许多情况下它非常方便：

- 程序员之间分享代码与项目
- 阅读以前的项目代码
- 重用函数

当脚本比较短时，注释看着比较琐碎，但是它应该从学习第一天开始积累。看下面的代码：

```
a = 10
b = op('value')[0]
op('op12').par.rx = b
c = str(op('numbers')[0])
d = 'testing'
e = 'something'
op('lines').par.text = d + e + c
```

这个脚本难以理解有多种原因，其中最大原因是它的作用不明确。一个快速增加脚本阅读效率的方法就是添加注释。让我们为上面的代码添加注释之后再来阅读：

```
#Set initial variable
#get the value from slider input
a = 10
b = op('value')[0]
#assign the slider value to video input rotation
op('op12').par.rx = b
#take numeric input from computer keypad
c = str(op('numbers')[0])
d = 'You'
e = 'pressed'
#create a sentence from variables above
#add it to a Text TOP to display
op('lines').par.text = d + e + c
```

不用花费时间思考创建有具体含义的变量与元件名，上面的代码仍然非常容易阅读。即使脱离上下文，这个脚本的功能也非常显而易见。

这种简单的添加注释可以使开发者之间的工作更轻松愉快。

9.5 划分

项目变得越来越复杂时，脚本也慢慢得变得越来越长。在这时 TD 将需要花费更多时间搜索代码，甚至于超过运行代码的时间。基于这些原因，脚本的划分对所有项目都非常重要。

这意味着我们需要运用不同的模块和技术去改善工作流程 ，而随之而来更大的好处是它们会消除项目合作间的痛苦。其他优点如：

- 易于长期维护脚本
- 花更少时间向同事解释脚本
- 增加程序的重用性
- 更快速的进行简单编辑和代码维护

打开“Scripting_1.toe”，在这个案例中，有 10 个 Movie In TOP 并执行各自的动作。首先，将每一个视频卸载。然后，为第一个 TOP 匹配一个文件路径。最后，为每个视频的播放作预载。因为这是一个 Python 案例，所有动作都使用 Python 脚本执行。下面查看“movies”Text DAT。

快速浏览一下脚本：出于案例演示需要，这个脚本没有使用循环遍历所有的操作元件。虽然只执行几个单独的动作，但这足以体会到操作一个又长又复杂的脚本是什么感觉。

在“movies”Text DAT 上点击右键，选择“Run Script”，所有的动作将依序运行。这个脚本已经添加了注释，但是仔细阅读代码仍会让你感到困惑。如果要在代码中编辑其中的某个值，不得不在长长的动作列表中查找。同事们将花费更多时间才能弄清脚本的作用。而在将来要重用脚本中的代码片段时，需要手动查找并提取。如何才能提升这个过程的效率呢？

打开“Scripting_2.toe”，这个案例使用上一案例的代码，但是将我们的动作按类独立分割在不同的 Text DAT 中。

即使不查看每个脚本，也可以快速的知道各个元件的功能：一个卸载视频，一个改变路径，一个预载视频。

在每一类脚本的结尾有一行代码将按序运行每一个过程的脚本。该行代码使用 Run 类。

op('preload').run()

这可以很轻易的编辑任意一个动作或参数值，并进行跟踪。向同事共享这个组件时，他们第一眼就可以看明白每个脚本的作用。如果有人需要启动时预载一组 Movies In TOP，也可以快速的从这个项目中提取脚本。

但这样的划分方法难以管理超过 500 行的脚本，分成更小的脚本易于管理与分享。在这种情况下，有第三种方法来控制多个元件的运行。

打开“Scripting_3.toe”，这个案例使用到 Python 函数。一个 Python 函数是一小段代码，可以被作为一系列动作进行调用。在“actions” Text DAT 中，定义一个函数，它包含每个 Movie In TOP 需要被执行的一系列动作。在“set_movie_tops” Text DAT 中而同样的动作不是被一遍又一遍地重复输入，而是每一个 Movie In TOP 都调用执行此函数。

虽然这一系列动作有些牵强，但是它的主旨是明确的：使 Python 脚本易于维护，在工作流程中易于使用，可重复利用，且方便管理。

9.6 外部模块

在 TouchDesigner 中集成的 Python 有尚未完全发挥的能力那就是它可以导入与使用 Python 的第三方库。

这包括使用当下比较流行的 Requests 库或 Beautiful Soup 库。导入库的过程非常简单：

1. 安装 64 位的 Python，在写此文章时的最新版本是 3.5。
2. 使用“Pip”安装要用到的外部模块。
3. 打开 TouchDesigner，“Edit”菜单下选择“Preferences”。
4. 在“General”下找到“Python 64-bit Module Path”参数，添加“site-packages”目录的路径。可以在 Python 安装目录中的“Lib”目录下找到。

5. 进入 TouchDesigner，创建一个 Text DAT，使用标准“import”命令进行测试。这个方法常用于外部模块的版本测试，用户经常遇到的问题是使用“Pip”安装外部模块虽然安装成功但版本却不是期望的。

9.7 哪里学习 Python

下面是一些在线免费学习 Python 的资源（注：有一些非常好的教程是使用 Python2.7，所以也一并包含在内）

CodeAcademy http://www.codecademy.com/tracks/python
Khan Academy https://www.khanacademy.org/science/computer-science
Coursera https://www.coursera.org/course/interactivepython
Google Developers https://developers.google.com/edu/python/?csw=1
Learn Python The Hard Way http://learnpythonthehardway.org/